Benchmarking Methodology Working Group
Internet-Draft
Intended status: Informational
Expires: 3 September 2022

K. Sun
ETRI
H. Yang
KT
J. Lee
T. Ngoc
Y. Kim
Soongsil University

March 2022

Considerations for Benchmarking Network Performance in Containerized Infrastructures
draft-dcn-bmwg-containerized-infra-08

Abstract

This draft describes considerations for benchmarking network performance in containerized infrastructures. In the containerized infrastructure, Virtualized Network Functions(VNFs) are deployed on an operating-system-level virtualization platform by abstracting the user namespace as opposed to virtualization using a hypervisor. Hence, the system configurations and networking scenarios for benchmarking will be partially changed by how the resource allocation and network technologies are specified for containerized VNFs. This draft compares the state of the art in the container networking architecture with VM-based virtualized systems networking architecture and provides several test scenarios for benchmarking network performance in containerized infrastructures.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Benchmarking Methodology Working Group(BMWG) has recently expanded its benchmarking scope from Physical Network Function(PNF) running on a dedicated hardware system to Network Function Virtualization(NFV) infrastructure and Virtualized Network Function(VNF). [RFC8172] described considerations for configuring NFV infrastructure and benchmarking metrics, and [RFC8204] gives guidelines for benchmarking virtual switch which connects VNFs in Open Platform for NFV(OPNFV).

Recently NFV infrastructure has evolved to include a lightweight virtualized platform called the containerized infrastructure, where VNFs share the same host Operating System(OS) and are logically isolated by using a different namespace. While previous NFV infrastructure uses a hypervisor to allocate resources for Virtual Machine(VMs) and instantiate VNFs, the containerized infrastructure virtualizes resources without a hypervisor, making containers very lightweight and more efficient in infrastructure resource utilization compared to the VM-based NFV infrastructure. When we consider benchmarking for VNFs in the containerized infrastructure, it may have a different System Under Test(SUT) and Device Under Test(DUT) configuration compared with both black-box benchmarking and VM-based NFV infrastructure as described in [RFC8172]. Accordingly, additional configuration parameters and testing strategies may be required.

In the containerized infrastructure, a VNF network is implemented by running both switch and router functions in the host system. For example, the internal communication between VNFs in the same host uses the L2 bridge function, while communication with external node(s) uses the L3 router function. For container networking, the host system may use a virtual switch(vSwitch), but other options exist. In the [ETSI-TST-009], they describe differences in networking structure between the VM-based and the containerized infrastructure. Occasioned by these differences, deployment scenarios for testing network performance described in [RFC8204] may be partially applied to the containerized infrastructure, but other scenarios may be required.

This draft aims to distinguish benchmarking of containerized infrastructure from the previous benchmarking methodology of common NFV infrastructure. Considering the point in [RFC8204] that virtual switch (vSwitch) is the networking principle of containerized infrastructure, this draft investigates different network models based on vSwitch location and acceleration technologies. At the same time, it is essential to uncover the impact of different deployment configurations on containerized infrastructure, such as resource

isolation, hugepages, service function chaining. The benchmark experiences of various combinations of these mentioned configurations and and networking models are also presented in this draft as the references to set up and benchmark containerized infrastructure. Note that, although the detailed configurations of both infrastructures differ, the new benchmarks and metrics defined in [RFC8172] can be equally applied in containerized infrastructure from a generic-NFV point of view, and therefore defining additional metrics or methodologies are out of scope.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document is to be interpreted as described in [RFC2119]. This document uses the terminology described in [RFC8172], [RFC8204], [ETSI-TST-009].

3. Containerized Infrastructure Overview

For benchmarking of the containerized infrastructure, as mentioned in [RFC8172], the basic approach is to reuse existing benchmarking methods developed within the BMWG. Various network function specifications defined in BMWG should still be applied to containerized VNF(C-VNF)s for the performance comparison with physical network functions and VM-based VNFs. A major distinction of the containerized infrastructure from the VM-based infrastructure is the absence of a hypervisor. Without hypervisor, all C- VNFs share the same host resources, including but not limited to computing, storage, and networking resources, as well as the host Operating System(OS), kernel, and libraries. These architectural differences bring additional considerations of resource management impacts for benchmarking.

In a common containerized infrastructure, thanks to the proliferation of Kubernetes, the pod is defined as a basic unit for orchestration and management that can host multiple containers. Based on that, [ETSI-TST-009] defined two test scenario for container infrastructure as follows.

o Container2Container: Communication between containers running in the same pod. it can be done by shared volumes or Inter-process communication (IPC).

o Pod2Pod: Communication between containers running in the different pods.

As mentioned in [RFC8204], vSwitch is also an important aspect of the containerized infrastructure. For Pod2Pod communication, every pod has only one virtual Ethernet (vETH) interface. This interface is connected to the vSwitch via vETH pair for each container. Not only Pod2Pod but also Pod2External scenario that communicates with an external node is also required. In this case, vSwitch SHOULD support gateway and Network Address Translation (NAT) functionalities.

Figure 1 shows briefly differences of network architectures based on container deployment models. Basically, on bare metal, C-VNFs can be deployed as a cluster called POD by Kubernetes. Otherwise, each C-VNF can be deployed separately using Docker. In the former case, there is only one external network interface, even a POD containing more than one C-VNF. An additional deployment model considers a scenario where C-VNFs or PODs are running on VM. In our draft, we define new terminologies; BMP, which is Pod on bare metal, and VMP, which is Pod on VM.

```
 +---------------------------------------------------------------------+
 |                          Baremetal Node                             |
 | +--------------+ +--------------+ +-------------- + +-------------+ |
 | |              | |     POD      | |      VM       | |     VM      | |
 | |              | |+------------+| |+-------------+| |  +-------+  | |
 | |   C-VNF(A)   | || C-VNFs(B)  || ||  C-VNFs(C)  || |  |PODs(D)|  | |
 | |              | |+------------+| |+-----^-------+| |  +---^---+  | |
 | |              | |              | |      |        | |      |      | |
 | |   +------+   | |   +------+   | |   +--v---+    | |  +---v--+   | |
 | +---| veth |---+ +---| veth |---+ +---|virtio|----+ +--|virtio|---+ |
 |     +--^---+         +---^--+         +--^---+         +---^--+     |
 |        |                 |               |                 |        |
 |        |                 |            +--v---+         +---v--+     |
 | +------|-----------------|------------|vhost |---------|vhost |---+ |
 | |      |                 |            +--^---+         +---^--+   | |
 | |      |                 |               |                 |      | |
 | |   +--v---+         +---v--+         +--v---+         +---v--+   | |
 | | +-| veth |---------| veth |---------| Tap  |---------| Tap  |-+ | |
 | | | +--^---+         +---^--+         +--^---+         +---^--+ | | |
 | | |      |                 |    vSwitch    |                 |     | | |
 | | | +--|-----------------|---------------|-----------------|--+ | | |
 | | +-|  |                 |    Bridge     |                 |  |-+ | |
 | |   +--|-----------------|---------------|-----------------|--+   | |
 | |      |   +---------+  |            +--|-----------------|---+  | |
 | |      |   |Container|  |            |  |    Hypervisor   |   |  | |
 | |      |   | Engine  |  |            |  |                 |   |  | |
 | |      |   +---------+  |            +--|-----------------|---+  | |
 | |      |                 |  Host Kernel  |                 |      | |
 | +------|-----------------|---------------|-----------------|------+ |
 |     +--v-----------------v---------------v-----------------v--+     |
 +-----|                  physical network                  |-----+
       +---------------------------------------------------------+
```

Figure 1: Examples of Networking Architecture based on Deployment Models - (A)C-VNF on Baremetal (B)Pod on Baremetal(BMP) (C)C-VNF on VM (D)Pod on VM(VMP)

In [ETSI-TST-009], they described data plane test scenarios in a single host. In that document, there are two scenarios for containerized infrastructure; Container2Container, which is internal communication between two containers in the same Pod, and the Pod2Pod model, which is communication between two containers running in different Pods. According to our new terminologies, we can call the Pod2Pod model the BMP2BMP scenario. When we consider container running on VM as an additional deployment option, there can be more single host test scenarios as follows;

o BMP2VMP scenario

```
 +---------------------------------------------------------------------+
 | HOST                                +-----------------------------+   |
 |                                   |VM +-------------------+     |   |
 |                                   |   |        C-VNF       |     |   |
 |  +--------------------+           |   | +--------------+   |     |   |
 |  |      C-VNF         |           |   | | Logical Port |   |     |   |
 |  | +--------------+   |           |   +-+--^-------^---+---+     |   |
 |  | | Logical Port |   |           |   +----|--------|---+         |   |
 |  +-+--^-------^---+---+           |   |  Logical Port  |        |   |
 |       |       |                   +---+----^-------^---+--------+   |
 |       |       |                            |       |                |
 |  +----v-------|----------------------------|-------v-------------+  |
 |  |            l----------------------------l                     |  |
 |  |                    Data Plane Networking                      |  |
 |  |                    (Kernel or User space)                     |  |
 |  +----^--------------------------------------------^-------------+  |
 |       |                                            |                |
 |  +----v------+                                +----v------+         |
 |  |  Phy Port |                                |  Phy Port |         |
 |  +------------+                                +------------+
 +-------^--------------------------------------------^----------------+
         |                                            |
 +-------v--------------------------------------------v----------------+
 |                                                                     |
 |                           Traffic Generator                         |
 |                                                                     |
 +---------------------------------------------------------------------+
```

Figure 2: Single Host Test Scenario - BMP2VMP

o VMP2VMP scenario

```
 +---------------------------------------------------------------------+
 |  HOST                                                               |
 |  +-----------------------------+   +-----------------------------+  |
 |  |VM +-------------------+     |   |VM +-------------------+     |  |
 |  |   |       C-VNF       |     |   |   |       C-VNF       |     |  |
 |  |   | +--------------+  |     |   |   | +--------------+  |     |  |
 |  |   | | Logical Port |  |     |   |   | | Logical Port |  |     |  |
 |  |   +-+--^-------^---+--+     |   |   +-+--^-------^---+--+     |  |
 |  |   +----|-------|---+        |   |   +----|-------|---+        |  |
 |  |   |  Logical Port  |        |   |   |  Logical Port  |        |  |
 |  +---+----^-------^---+--------+   +---+----^-------^---+--------+  |
 |           |       |                        |       |                |
 |  +--------v-------v------------------------|-------v-------------+  |
 |  |                l------------------------l                     |  |
 |  |                    Data Plane Networking                      |  |
 |  |                    (Kernel or User space)                     |  |
 |  +----^--------------------------------------------^-------------+  |
 |       |                                            |                |
 |  +----v------+                                +----v------+         |
 |  |  Phy Port |                                |  Phy Port |         |
 |  +-----------+                                +-----------+         |
 +-------^--------------------------------------------^----------------+
         |                                            |
 +-------v--------------------------------------------v----------------+
 |                                                                     |
 |                           Traffic Generator                         |
 |                                                                     |
 +---------------------------------------------------------------------+
```

Figure 3: Single Host Test Scenario - VMP2VMP

4. Networking Models in Containerized Infrastructure

Container networking services are provided as network plugins. Basically, by using them, network services are deployed as an isolation environment from container runtime through the host namespace, creating a virtual interface, allocating interface and IP address to C-VNF. Since the containerized infrastructure has different network architecture depending on its using plugins, it is necessary to specify the plugin used in the infrastructure. Especially for Kubernetes infrastructure, several Container Networking Interface (CNI) plugins are developed, which describes network configuration files in JSON format, and plugins are instantiated as new namespaces. When the CNI plugin is initiated, it pushes forwarding rules and networking policies to the existing vSwitch (i.e., Linux bridge, Open vSwitch) or creates its own switch functions to provide networking service.

The container network model can be classified according to the location of the vSwitch component. There are some CNI plugins that provide networking without the vSwitch components; however, this draft focuses on plugins using vSwitch components.

4.1. Kernel-space vSwitch Model

```
    +------------------------------------------------------------------+
    | User Space                                                       |
    |   +-----------+                                  +-----------+   |
    |   |   C-VNF   |                                  |   C-VNF   |   |
    |   | +-------+ |                                  | +-------+ |   |
    |   +-|  eth  |-+                                  +-|  eth  |-+   |
    |     +---^---+                                      +---^---+     |
    |         |                                              |         |
    |         |     +----------------------------------+     |         |
    |         |     |                                  |     |         |
    |         |     |  Networking Controller / Agent   |     |         |
    |         |     |                                  |     |         |
    |         |     +-----------------^^---------------+     |         |
    ----------|-----------------------||---------------------|----------
    |     +---v---+                   ||                 +---v---+     |
    |  +--|  veth |-------------------vv-----------------|  veth |--+  |
    |  |  +-------+          vSwitch Component           +-------+  |  |
    |  |           (OVS Kernel Datapath, Linux Bridge, ..)          |  |
    |  |                                                            |  |
    |  +-------------------------------^----------------------------+  |
    |                                  |                               |
    | Kernel Space         +-----------v----------+                    |
    +----------------------|          NIC         |--------------------+
                           +----------------------+
```

Figure 4: Examples of Kernel-Space vSwitch Model

Figure 4 shows kernel-space vSwitch model. In this model, because the vSwitch component is running on kernel space, data packets should be processed in-network stack of host kernel before transferring packets to the C-VNF running in user-space. Not only pod2External but also pod2pod traffic should be processed in the kernel space. For dynamic networking configuration, the Forwarding policy can be pushed by the controller/agent located in the user-space. In the case of Open vSwitch (OVS) [OVS], the first packet of flow can be sent to the user space agent (ovs-switchd) for forwarding decision. Kernel-space vSwitch models are listed below;

o Docker Network[Docker-network], Flannel Network[Flannel], OVS(OpenvSwitch)[OVS], OVN(Open Virtual Network)[OVN]

4.2. User-space vSwitch Model

```
    +------------------------------------------------------------------+
    | User Space                                                       |
    |   +---------------+                          +---------------+   |
    |   |     C-VNF     |                          |     C-VNF     |   |
    |   | +-----------+ |    +-----------------+   | +-----------+ |   |
    |   | |virtio-user| |    |    Networking   |   | |virtio-user|-|   |
    |   +-|   / eth   |-+    | Controller/Agent|   +-|   / eth   |-+   |
    |     +-----^-----+      +-------^^--------+     +-----^-----+     |
    |           |                    ||                    |           |
    |           |                    ||                    |           |
    |     +-----v-----+              ||              +-----v-----+     |
    |     | vhost-user|              ||              | vhost-user|     |
    |  +--|  / memif  |--------------vv--------------|  / memif  |--+  |
    |  |  +-----------+                              +-----------+  |  |
    |  |                          vSwtich                           |  |
    |  |                      +--------------+                      |  |
    |  +----------------------|  PMD Driver  |----------------------+  |
    |                         |              |                         |
    |                         +-------^------+                         |
    ----------------------------------|---------------------------------
    |                                 |                                |
    |                                 |                                |
    |                                 |                                |
    | Kernel Space         +----------V-----------+                    |
    +----------------------|          NIC         |--------------------+
                           +----------------------+
```

Figure 5: Examples of User-Space vSwitch Model

Figure 5 shows user-space vSwitch model, in which data packets from physical network port are bypassed kernel processing and delivered directly to the vSwitch running on user-space. This model is commonly considered as Data Plane Acceleration (DPA) technology since it can achieve high-rate packet processing than a kernel-space network with limited packet throughput. For bypassing kernel and directly transferring the packet to vSwitch, Data Plane Development

Kit (DPDK) is essentially required. With DPDK, an additional driver called Pull-Mode Driver (PMD) is created on vSwtich. PMD driver must be created for each NIC separately. User-space vSwitch models are listed below;

- ovs-dpdk[ovs-dpdk], vpp[vpp]

4.3. eBPF Acceleration Model

```
    +------------------------------------------------------------------+
    | User Space                                                       |
    |    +----------------+                     +----------------+     |
    |    |      C-VNF     |                     |      C-VNF     |     |
    |    | +------------+ |                     | +------------+ |     |
    |    +-|    veth    |-+                     +-|    veth    |-+     |
    |      +-----^------+                         +------^-----+       |
    |            |                                       |             |
    -------------|---------------------------------------|--------------
    |      +-----v------+                         +------v-----+       |
    |      |    veth    |                         |    veth    |       |
    |      +-----^------+                         +------^-----+       |
    |            |                                       |             |
    |      +-----v------+                         +------v-----+       |
    |      |  tc eBPF   |                         |  tc eBPF   |       |
    |      |   Egress   |                         |   Egress   |       |
    |      +-----^------+                         +------^-----+       |
    |            |                                       |             |
    |            +--------------+          +-------------+             |
    |                           |          |                           |
    |                         +-v----------v-+                         |
    |                         |   tc eBPF    |                         |
    |                         |   Ingress    |                         |
    |                         +------^-------+                         |
    |                                |                                 |
    |                         +------v-------+                         |
    |                         |     XDP      |                         |
    |                         +------^-------+                         |
    |                                |                                 |
    | Kernel Space          +--------v--------+                        |
    +-----------------------|       NIC       |------------------------+
                            +-----------------+
```

Figure 6: Examples of eBPF Acceleration Model

Figure 6 shows eBPF Acceleration model, which leverages extended Berkeley Packet Filter (eBPF) technology [eBPF] to achieve high-performance packet processing. It enables execution of sandboxed programs inside abstract virtual machines within the Linux kernel without changing the kernel source code or loading the kernel module. To accelerate data plane performance, eBPF programs are attached to different BPF hooks inside the linux kernel stack.

One type of BPF hook is the eXpress Data Path (XDP) at the networking driver. It is the first hook that triggers eBPF program upon packet reception from external network. The other type of BPF hook is Traffic Control Ingress/Egress eBPF hook (tc eBPF). These hooks are attached to the vETH pair of the pod and the XDP hook. The tc Egress

eBPF hooks at the vETH pair enforce policy on all traffic exit the pod, while the tc Ingress eBPF hook at the end of the kernel networking runs after initial packet processing from XDP hook.

On the egress datapath side, whenever a packet exits the pod, it goes through vETH pair then is picked up by the tc egress eBPF hook. These hooks trigger eBPF programs to forward the packet directly to the external facing network interface, bypassing all of the kernel network layer processing such as iptables. On the ingress datapath side, eBPF programs at the XDP and tc ingress eBPF hook pick up packets from the network device and directly deliver it to the vETH interface pair, or bypassing context-switching process to the pod network namespace in the case of Cilium project [Cilium].

Notable eBPF Acceleration models are 2 CNI plugin projects: Calico[Calico], Cilium[Cilium]. In the case of Cilium, eBPF/XDP program can be offloaded directly on the smart NIC card, which allows data plane acceleration without using the CPU. Container network performance of these eBPF-based project is reported in [cilium-benchmark].

4.4. Smart-NIC Acceleration Model

```
 +------------------------------------------------------------------+
 | User Space                                                       |
 |    +-----------------+                    +-----------------+    |
 |    |      C-VNF      |                    |      C-VNF      |    |
 |    | +-------------+ |                    | +-------------+ |    |
 |    +-|  vf driver  |-+                    +-|  vf driver  |-+    |
 |      +-----^-------+                        +------^------+      |
 |            |                                       |             |
 -------------|---------------------------------------|---------------
 |            +---------+                   +---------+             |
 |               +------|-------------------|------+                |
 |               | +----v-----+       +-----v----+ |                |
 |               | | virtual  |       | virtual  | |                |
 |               | | function |       | function | |                |
 | Kernel Space  | +----^-----+  NIC  +-----^----+ |                |
 +---------------|      |                   |      |----------------+
                 | +----v-------------------v----+ |
                 | |      Classify and Queue     | |
                 | +-----------------------------+ |
                 +---------------------------------+
```

Figure 7: Examples of Smart-NIC Acceleration Model

Figure 7 shows Smart-NIC acceleration model, which does not use vSwitch component. This model can be separated into two technologies.

One is Single-Root I/O Virtualization (SR-IOV)[SR-IOV], which is an extension of PCIe specifications to enable multiple partitions running simultaneously within a system to share PCIe devices. In the NIC, there are virtual replicas of PCI functions known as virtual functions (VF), and each of them is directly connected to each container's network interfaces. Using SR-IOV, data packets from external bypass both kernel and user space and are directly forwarded to container's virtual network interface.

The other technology is eBPF/XDP programs offloading to Smart-NIC card as mentioned in the previous section. It enables general acceleration of eBPF. eBPF programs are attached to XDP and run at the Smart-NIC card, which allows server CPUs to perform more application-level work. However, not all Smart-NIC cards provide eBPF/XDP offloading support.

4.5. Model Combination

```
     +-------------------------------------------------------+
     | User Space                                            |
     | +--------------------+         +--------------------+ |
     | |       C-VNF        |         |        C-VNF       | |
     | | +------+  +------+ |         | +------+  +------+ | |
     | +-| veth |--| veth |-+         +-| veth |--| veth |-+ |
     |   +---^--+  +---^--+             +--^---+  +---^--+   |
     |       |         |                   |          |      |
     |       |         |                   |          |      |
     |       |     +---v--------+  +-------v----+     |      |
     |       |     | vhost-user |  | vhost-user |     |      |
     |       |  +--|  / memif   |--|  / memif   |--+  |      |
     |       |  |  +------------+  +------------+  |  |      |
     |       |  |             vSwitch              |  |      |
     |       |  +----------------------------------+  |      |
     |       |                                        |      |
     --------|----------------------------------------|-------
     |       +-----------+              +-------------+      |
     |              +----|--------------|---+                |
     |              |+---v--+       +---v--+|                |
     |              ||  vf  |       |  vf  ||                |
     |              |+------+       +------+|                |
     | Kernel Space |                       |                |
     +--------------|           NIC         |----------------+
                    +-----------------------+
```

Figure 8: Examples of Model Combination deployment

Figure 8 shows the networking model when combining user-space vSwitch model and Smart-NIC acceleration model. This model is frequently considered in service function chain scenarios when two different types of traffic flows are present. These two types are North/South traffic and East/West traffic.

North/South traffic is the type that packets are received from other servers and routed through VNF. For this traffic type, Smart-NIC model such as SR-IOV is preferred because packets always have to pass the NIC. User-space vSwitch involvement in north-south traffic will create more bottlenecks. On the other hand, East/West traffic is a form of sending and receiving data between containers deployed in the same server and can pass through multiple containers. For this type, user-space vSwitch models such as OVS-DPDK and VPP are preferred because packets are routed within the user space only and not through the NIC.

The throughput advantages of these different networking models with different traffic direction cases are reported in [Intel-SRIOV-NFV].

5. Performance Impacts

5.1. CPU Isolation / NUMA Affinity

CPU pinning enables benefits such as maximizing cache utilization, eliminating operating system thread scheduling overhead as well as coordinating network I/O by guaranteeing resources. This technology is very effective in avoiding the "noisy neighbor" problem, and it is already proved in existing experience [Intel-EPA].

Using NUMA, performance will be increasing not CPU and memory but also network since that network interface connected PCIe slot of specific NUMA node have locality. Using NUMA requires a strong understanding of VNF's memory requirements. If VNF uses more memory than a single NUMA node contains, the overhead will occurr due to being spilled to another NUMA node. Network performance can be changed depending on the location of the NUMA node whether it is the same NUMA node where the physical network interface and CNF are attached to. There is benchmarking experience for cross-NUMA performance impacts [ViNePERF]. In that tests, they consist of cross-NUMA performance with 3 scenarios depending on the location of the traffic generator and traffic endpoint. As the results, it was verified as below:

o A single NUMA Node serving multiple interfaces is worse than Cross-NUMA Node performance degradation

o Worse performance with VNF sharing CPUs across NUMA

5.2. Hugepages

Hugepage configures a large page size of memory to reduce Translation Lookaside Buffer(TLB) miss rate and increase the application performance. This increases the performance of logical/virtual to physical address lookups performed by a CPU's memory management unit, and overall system performance. In the containerized infrastructure, the container is isolated at the application level, and administrators can set huge pages more granular level (e.g., Kubernetes allows to use of 512M bytes huge pages for the container as default values). Moreover, this page is dedicated to the application but another process, so the application uses the page more efficiently way. From a network benchmark point of view, however, the impact on general packet processing can be relatively negligible, and it may be necessary to consider the application level to measure the impact together. In the case of using the DPDK application, as reported in [Intel-EPA], it was verified to improve network performance because packet handling processes are running in the application together.

5.3. Service Function Chaining

When we consider benchmarking for containerized and VM-based infrastructure and network functions, benchmarking scenarios may contain various operational use cases. Traditional black-box benchmarking focuses on measuring the in-out performance of packets from physical network ports since the hardware is tightly coupled with its function and only a single function is running on its dedicated hardware. However, in the NFV environment, the physical network port commonly will be connected to multiple VNFs(i.e., Multiple PVP test setup architectures were described in [ETSI-TST-009]) rather than dedicated to a single VNF. This scenario is called Service Function Chaining. Therefore, benchmarking scenarios should reflect operational considerations such as the number of VNFs or network services defined by a set of VNFs in a single host. [service-density] proposed a way for measuring the performance of multiple NFV service instances at a varied service density on a single host, which is one example of these operational benchmarking aspects. Another aspect in benchmarking service function chaining scenario should be considered is different network acceleration technologies. Network performance differences may occur because of different traffic patterns based on the provided acceleration method.

5.4. Additional Considerations

Apart from the single-host test scenario, the multi-hosts scenario should also be considered in container network benchmarking, where container services are deployed across different servers. To provide network connectivity for container-based VNFs between different server nodes, inter-node networking is required. According to [ETSI-NFV-IFA-038], there are several technologies to enable inter-node network: overlay technologies using a tunnel endpoint (e.g. VXLAN, IP in IP), routing using Border Gateway Protocol (BGP), layer 2 underlay, direct network using dedicated NIC for each pod, or load balancer using LoadBalancer service type in Kubernetes. Different protocols from these technologies may cause performance differences in container networking.

6. Security Considerations

TBD

7. References

7.1. Informative References

[Calico] "Project Calico", July 2019, <https://docs.projectcalico.org/>.

[Cilium] "Cilium Documentation", March 2022, <https://docs.cilium.io/en/stable//>.

[cilium-benchmark] Cilium, "CNI Benchmark: Understanding Cilium Network Performance", May 2021, <https://cilium.io/blog/2021/05/11/cni-benchmark>.

[Docker-network] "Docker, Libnetwork design", July 2019, <https://github.com/docker/libnetwork/>.

[DPDK_eBPF] "DPDK-Berkeley Packet Filter Library", August 2021, <https://doc.dpdk.org/guides/prog_guide/bpf_lib.html>.

[eBPF] "eBPF, extended Berkeley Packet Filter", July 2019, <https://www.iovisor.org/technology/ebpf>.

[ETSI-NFV-IFA-038]
"Network Functions Virtualisation (NFV) Release 4; Architectural Framework; Report on network connectivity for container-based VNF", November 2021.

[ETSI-TST-009]
"Network Functions Virtualisation (NFV) Release 3; Testing; Specification of Networking Benchmarks and Measurement Methods for NFVI", October 2018.

[Flannel] "flannel 0.10.0 Documentation", July 2019, <https://coreos.com/flannel/>.

[Intel-EPA]
Intel, "Enhanced Platform Awareness in Kubernetes", 2018, <https://builders.intel.com/docs/networkbuilders/enhanced-platform-awareness-feature-brief.pdf>.

[Intel-SRIOV-NFV]
Patrick, K. and J. Brian, "SR-IOV for NFV Solutions Practical Considerations and Thoughts", February 2017.

[OVN] "How to use Open Virtual Networking with Kubernetes", July 2019, <https://github.com/ovn-org/ovn-kubernetes>.

[OVS] "Open Virtual Switch", July 2019, <https://www.openvswitch.org/>.

[ovs-dpdk] "Open vSwitch with DPDK", July 2019, <http://docs.openvswitch.org/en/latest/intro/install/dpdk/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8172] Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", RFC 8172, July 2017, <https://www.rfc-editor.org/rfc/rfc8172>.

[RFC8204] Tahhan, M., O'Mahony, B., and A. Morton, "Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV)", RFC 8204, September 2017, <https://www.rfc-editor.org/rfc/rfc8204>.

[service-density]
Konstantynowicz, M. and P. Mikus, "NFV Service Density Benchmarking", March 2019, <https://tools.ietf.org/html/draft-mkonstan-nf-service-density-00>.

[SR-IOV] "SRIOV for Container-networking", July 2019, <https://github.com/intel/sriov-cni>.

[userspace-cni]
"Userspace CNI Plugin", August 2021, <https://github.com/intel/userspace-cni-network-plugin>.

[ViNePERF] Anuket Project, "Cross-NUMA performance measurements with VSPERF", March 2019, <https://wiki.anuket.io/display/HOME/Cross-NUMA+performance+measurements+with+VSPERF>.

[vpp] "VPP with Containers", July 2019, <https://fdio-vpp.readthedocs.io/en/latest/usecases/containers.html>.

Appendix A. Benchmarking Experience(Contiv-VPP)

A.1. Benchmarking Environment

In this test, our purpose is to test the performance of user-space based model for container infrastructure and figure out the relationship between resource allocation and network performance. With respect to this, we set up Contiv-VPP, one of the user-space based network solutions in container infrastructure and tested like below.

o Three physical server for benchmarking

Node Name	Specification	Description
Conatiner Control for Master	- Intel(R) Xeon(R) CPU E5-2690 (2Socket X 12Core) - MEM 128G - DISK 2T - Control plane : 1G	Container Deployment and Network Allocation - ubuntu 18.04 - Kubernetes Master - CNI Conterller .. Contive VPP Controller .. Contive VPP Agent
Conatiner Service for Worker	- Intel(R) Xeon(R) Gold 6148 (2socket X 20Core) - MEM 128G - DISK 2T - Control plane : 1G - Data plane : MLX 10G (1NIC 2PORT)	Container Service - ubuntu 18.04 - Kubernetes Worker - CNI Agent .. Contive VPP Agent
Packet Generator	- Intel(R) Xeon(R) CPU E5-2690 (2Socket X 12Core) - MEM 128G - DISK 2T - Control plane : 1G	Packet Generator - CentOS 7 - installed Trex 2.4

```
 |                   |- Data plane : MLX 10G|                          |
 |                   |  (1NIC 2PORT)        |                          |
 +-------------------+----------------------+--------------------------+
```

Figure 9: Test Environment-Server Specification

o The architecture of benchmarking

```
     +----+   +--------------------------------------------------------+
     |    |   |  Containerized Infrastructure Master Node              |
     |    |   |  +-----------+                                         |
     |   <-------> 1G PORT 0 |                                         |
     |    |   |  +-----------+                                         |
     |    |   +--------------------------------------------------------+
     |    |
     |    |   +--------------------------------------------------------+
     |    |   |  Containerized Infrastructure Worker Node              |
     |    |   |                    +---------------------------------+ |
     | s  |   |  +-----------+     | +------------+   +------------+ | |
     | w <-------> 1G PORT 0 |     | | 10G PORT 0 |   | 10G PORT 1 | | |
     | i  |   |  +-----------+     | +------^-----+   +------^-----+ | |
     | t  |   |                    +--------|----------------|-------+ |
     | c  |   +-----------------------------|----------------|---------+
     | h  |                                 |                |
     |    |   +-----------------------------|----------------|---------+
     |    |   |  Packet Generator Node      |                |         |
     |    |   |                    +--------|----------------|-------+ |
     |    |   |  +-----------+     | +------v-----+   +------v-----+ | |
     |   <-------> 1G PORT 0 |     | | 10G PORT 0 |   | 10G PORT 1 | | |
     |    |   |  +-----------+     | +------------+   +------------+ | |
     |    |   |                    +---------------------------------+ |
     |    |   |                                                        |
     +----+   +--------------------------------------------------------+
```

Figure 10: Test Environment-Architecture

o Network model of Containerized Infrastructure(User space Model)

```
   +---------------------------------------------+---------------------+
   |                   NUMA 0                    |        NUMA 0       |
   +---------------------------------------------|---------------------+
   |  Containerized Infrastructure Worker Node   |                     |
   |        +---------------------------+        |  +----------------+ |
   |        |           POD1            |        |  |     POD2       | |
   |        |      +-------------+      |        |  |   +-------+    | |
   |        |      |             |      |        |  |   |       |    | |
   |        |   +--v---+     +---v--+   |        |  | +-v--+  +-v--+ | |
   |        |   | eth1 |     | eth2 |   |        |  | |eth1|  |eth2| | |
   |        |   +--^---+     +---^--+   |        |  | +-^--+  +-^--+ | |
   |        +------|-------------|------+        |  +---|-------|----+ |
   |            +---             |               |      |       |      |
   |            |        +-------|---------------|------+       |      |
   |            |        |       |        +------|--------------+      |
   | +----------|--------|-------|--------|----+ |                     |
   | |          v        v       v        v    | |                     |
   | |       +-tap10--tap11-+ +-tap20--tap21-+ | |                     |
   | |       |  ^        ^  | |  ^        ^  | | |                     |
   | |       |  |  VRF1  |  | |  |  VRF2  |  | | |                     |
   | |       +--|--------|--+ +--|--------|--+ | |                     |
   | |          |  +------+      |    +----+   | |                     |
   | | +-tap01--|--|-------------|----|---+    | |                     |
   | | | +------v--v-+ VRF0 +----v----v-+ |    | |                     |
   | | +-| 10G ETH0/0|------| 10G ETH0/1|-+    | |                     |
   | |   +---^-------+      +-------^---+      | |                     |
   | |   +---v-------+      +-------v---+      | |                     |
   | +---| DPDK PMD0 |------| DPDK PMD1 |------+ |                     |
   |     +---^-------+      +-------^---+        | User Space          |
   +---------|----------------------|------------|---------------------+
   |   +-----|----------------------|-----+      | Kernal Space        |
   +---| +---V----+            +----v---+ |------|---------------------+
       | | PORT 0 |  10G NIC    | PORT 1 | |      |
       | +---^----+            +----^---+ |
       +-----|----------------------|-----+
       +-----|----------------------|-----+
   +---| +---V----+            +----v---+ |----------------------------+
   |   | | PORT 0 |  10G NIC    | PORT 1 | |   Packet Generator (Trex)  |
   |   | +--------+            +--------+ |                            |
   |   +----------------------------------+                            |
   +-------------------------------------------------------------------+
```

Figure 11: Test Environment-Network Architecture

We set up a Contive-VPP network to benchmark the user space container network model in the containerized infrastructure worker node. We set up network interface at NUMA0, and we created different network subnets VRF1, VRF2 to classify input and output data traffic,

respectively. And then, we assigned two interfaces which connected to VRF1, VRF2 and, we setup routing table to route Trex packet from eth1 interface to eth2 interface in POD.

A.2. Trouble shooting and Result

In this environment, we confirmed that the routing table doesn't work when we send packets using Trex packet generator. The reason is that when kernel space based network configured, ip forwarding rule is processed to kernel stack level while 'ip packet forwarding rule' is processed only in vrf0, which is the default virtual routing and forwarding (VRF0) in VPP. The above testing architecture makes problem since vrf1 and vrf2 interface couldn't route packet. According to above result, we assigned vrf0 and vrf1 to POD and, data flow is like below.

```
   +---------------------------------------------+---------------------+
   |                   NUMA 0                    |        NUMA 0       |
   +---------------------------------------------|---------------------+
   |  Containerized Infrastructure Worker Node   |                     |
   |        +---------------------------+        |  +----------------+ |
   |        |      POD1                 |        |  |     POD2       | |
   |        |      +-------------+      |        |  |   +-------+    | |
   |        |   +--v----+    +---v--+   |        |  | +-v--+  +-v--+ | |
   |        |   | eth1 |     | eth2 |   |        |  | |eth1|  |eth2| | |
   |        |   +--^---+     +---^--+   |        |  | +-^--+  +-^--+ | |
   |        +------|-------------|------+        |  +---|-------|----+ |
   |       +-------+             |               |      |       |      |
   |       |       +-------------|---------------|------+       |      |
   |       |       |             |        +------|--------------+      |
   | +-----|-------|-------------|--------|----+ |                     |
   | |     |       |             v        v    | |                     |
   | |     |       |          +-tap10--tap11-+ | |                     |
   | |     |       |          |  ^        ^  | | |                     |
   | |     |       |          |  |  VRF1  |  | | |                     |
   | |     |       |          +--|--------|--+ | |                     |
   | |     |       |             |    +---+    | |                     |
   | | +-*tap00--*tap01----------|----|---+    | |                     |
   | | | +-V-------v-+ VRF0 +----v----v-+ |    | |                     |
   | | +-| 10G ETH0/0|------| 10G ETH0/1|-+    | |                     |
   | |   +-----^-----+      +------^----+      | |                     |
   | |   +-----v-----+      +------v----+      | |                     |
   | +---|*DPDK PMD0 |------|*DPDK PMD1 |------+ |                     |
   |     +-----^-----+      +------^----+        | User Space          |
   +-----------|-------------------|-------------|---------------------+
               v                   v
  *- CPU pinning interface
```

Figure 12: Test Environment-Network Architecture(CPU Pinning)

We conducted benchmarking with three conditions. The test environments are as follows. - Basic VPP switch - General kubernetes (No CPU Pining) - Shared Mode / Exclusive mode. In the basic Kubernetes environment, all PODs share a host's CPU. Shared mode is that some POD share a pool of CPU assigned to specific PODs. Exclusive mode is that a specific POD dedicates a specific CPU to use. In shared mode, we assigned two CPUs for several PODs, in exclusive mode, we dedicated one CPU for one POD, independently. The result is like Figure 13. First, the test was conducted to figure out the line rate of the VPP switch, and the basic Kubernetes performance. After that, we applied NUMA to the network interface using Shared Mode and Exclusive Mode in the same node and different node. In Exclusive and Shared mode tests, we confirmed that Exclusive mode showed better performance than Shared mode when same NUMA CPU was assigned, respectively. However, we confirmed that performance is reduced at the section between the vpp switch and the POD, affecting the total result.

Model	NUMA Mode (pinning)	Result(Gbps)
Maximum Line Rate	N/A	3.1
	same NUMA	9.8
Without CMK	N/A	1.5
CMK-Exclusive Mode	same NUMA	4.7
	Different NUMA	3.1
CMK-shared Mode	same NUMA	3.5
	Different NUMA	2.3

Figure 13: Test Results

Appendix B. Benchmarking Experience(SR-IOV with DPDK)

B.1. Benchmarking Environment

In this test, our purpose is to test the performance of Smart-NIC acceleration model for container infrastructure and figure out relationship between resource allocation and network performance. With respect to this, we setup SRIOV combining with DPDK to bypass the Kernel space in container infrastructure and tested based on that.

o Three physical server for benchmarking

Node Name	Specification	Description
Conatiner Control for Master	- Intel(R) Core(TM) i5-6200U CPU (1socket x 4Core) - MEM 8G - DISK 500GB - Control plane : 1G	Container Deployment and Network Allocation - ubuntu 18.04 - Kubernetes Master - CNI Conterller MULTUS CNI SRIOV plugin with DPDK
Conatiner Service for Worker	- Intel(R) Xeon(R) E5-2620 v3 @ 2.4Ghz (1socket X 6Core) - MEM 128G - DISK 2T - Control plane : 1G - Data plane : XL710-qda2 (1NIC 2PORT- 40Gb)	Container Service - Centos 7.7 - Kubernetes Worker - CNI Agent MULTUS CNI SRIOV plugin with DPDK
Packet Generator	- Intel(R) Xeon(R) Gold 6148 @ 2.4Ghz (2Socket X 20Core) - MEM 128G - DISK 2T - Control plane : 1G - Data plane : XL710-qda2 (1NIC 2PORT- 40Gb)	Packet Generator - CentOS 7.7 - installed Trex 2.4

Figure 14: Test Environment-Server Specification

   o The architecture of benchmarking

```
     +----+   +--------------------------------------------------------+
     |    |   |  Containerized Infrastructure Master Node              |
     |    |   |  +-----------+                                         |
     |   <-------> 1G PORT 0 |                                         |
     |    |   |  +-----------+                                         |
     |    |   +--------------------------------------------------------+
     |    |
     |    |   +--------------------------------------------------------+
     |    |   |  Containerized Infrastructure Worker Node              |
     |    |   |                    +---------------------------------+ |
     | s  |   |  +-----------+     | +------------+   +------------+ | |
     | w <-------> 1G PORT 0 |     | | 40G PORT 0 |   | 40G PORT 1 | | |
     | i  |   |  +-----------+     | +------^-----+   +------^-----+ | |
     | t  |   |                    +--------|----------------|-------+ |
     | c  |   +-----------------------------|----------------|---------+
     | h  |                                 |                |
     |    |   +-----------------------------|----------------|---------+
     |    |   |  Packet Generator Node      |                |         |
     |    |   |                    +--------|----------------|-------+ |
     |    |   |  +-----------+     | +------v-----+   +------v-----+ | |
     |   <-------> 1G PORT 0 |     | | 40G PORT 0 |   | 40G PORT 1 | | |
     |    |   |  +-----------+     | +------------+   +------------+ | |
     |    |   |                    +---------------------------------+ |
     |    |   |                                                        |
     +----+   +--------------------------------------------------------+
```

Figure 15: Test Environment-Architecture

   o Network model of Containerized Infrastructure(User space Model)

```
   +---------------------------------------------+---------------------+
   |             CMK shared core                 | CMK exclusive core  |
   +---------------------------------------------|---------------------+
   |  Containerized Infrastructure Worker Node   |                     |
   |        +---------------------------+        |  +----------------+ |
   |        |           POD1            |        |  |     POD2       | |
   |        |         (testpmd)         |        |  |   (testpmd)    | |
   |        |      +-------------+      |        |  |   +-------+    | |
   |        |      |             |      |        |  |   |       |    | |
   |        |   +--v---+     +---v--+   |        |  | +-v--+  +-v--+ | |
   |        |   | eth1 |     | eth2 |   |        |  | |eth1|  |eth2| | |
   |        |   +--^---+     +---^--+   |        |  | +-^--+  +-^--+ | |
   |        +------|-------------|------+        |  +---|-------|----+ |
   |               |             |               |      |       |      |
   |         +------           +-+               |      |       |      |
   |         |            +----|-----------------|------+       |      |
   |         |            |    |        +--------|--------------+      |
   |         |            |    |        |        |           User Space|
   +---------|------------|----|--------|--------|---------------------+
   |         |            |    |        |        |                     |
   |      +--+     +------|    |        |        |                     |
   |      |        |           |        |        |         Kernal Space|
   +------|--------|-----------|--------|--------+---------------------+
   | +----|--------|-----------|--------|-----+  |                     |
   | | +--v--+  +--v--+     +--v--+  +--v--+  |  |                  NIC|
   | | | VF0 |  | VF1 |     | VF2 |  | VF3 |  |  |                     |
   | | +--|---+ +|----+     +----|+  +-|---+  |  |                     |
   | +----|------|---------------|-----|------+  |                     |
   +---| +v------v+            +-v-----v+ |------|----------------------+
       | | PORT 0 |  40G NIC   | PORT 1 | |
       | +---^-----+           +-----^---+ |
       +-----|----------------------|-----+
       +-----|----------------------|-----+
   +---| +---V----+            +----v---+ |-----------------------------+
   |   | | PORT 0 |  40G NIC   | PORT 1 | |   Packet Generator (Trex)   |
   |   | +--------+            +--------+ |                             |
   |   +----------------------------------+                             |
   +--------------------------------------------------------------------+
```

Figure 16: Test Environment-Network Architecture

We set up a Multus CNI, SRIOV CNI with DPDK to benchmark the user-space container network model in the containerized infrastructure worker node. The Multus CNI support creates multiple interfaces for a container. The traffic is bypassed the Kernel space by SRIOV with DPDK. We established two modes of CMK: shared core and exclusive core. We created VFs for each network interface of a container. Then, we set up TREX to route packet from eth1 to eth2 in a POD.

B.2. Trouble shooting and Results

Figure 17 shows the test results when using 1518 bytes packet traffic from the T-Rex traffic generator. First, we get the maximum line rate of the system using SR-IOV as the packet acceleration technique. Then we measured throughput when applying the CMK feature. We observed similar results as VPP CPU Pinning test. The default Kubernetes system without CMK feature enabled had the worst performance as the CPU resources are shared without any isolation. When the CMK feature is enabled, Exclusive Mode performed better than Shared Mode because each pod had its own dedicated CPU.

Model	Result(Gbps)
Maximum Line Rate	39.3
Without CMK	11.5
CMK-Exclusive Mode	39.2

CMK-shared Mode	29.6

Figure 17: SR-IOV CPU Pinning Test Results

Appendix C. Benchmarking Experience(Multi-pod Test)

C.1. Benchmarking Overview

The main goal of this experience was to benchmark the multi-pod scenario, in which packets are traversed through two pods. To create additional interfaces for forwarding packets between two pods, Multus CNI was used. We compared two userspace-vSwitch model network technologies: OVS/DPDK and VPP-memif. Since that vpp-memif has a different packet forwarding mechanism by using shared memory interface, it is expected that vpp-memif may provide higher performance that OVS-DPDK. Also, we consider NUMA impact for both cases, and made 6 scenarios depending on CPU location of vSwitch and two pods. Figure 18 is packet forwarding scenario in this test, where two pods run on the same host and vSwitch delivers packets between two pods.

```
+----------------------------------------------------------------+
|Worker Node                                                     |
|   +--------------------------------------------------------+   |
|   |Kubernetes                                              |   |
|   |   +--------------+                +--------------+     |   |
|   |   |     pod1     |                |     pod2     |     |   |
|   |   |  +--------+  |                |  +--------+  |     |   |
|   |   |  |  L2FWD |  |                |  |  L2FWD |  |     |   |
|   |   |  +---^--v-+  |                |  +--^--v--+  |     |   |
|   |   |  |  DPDK  |  |                |  |  DPDK  |  |     |   |
|   |   |  +---^--v-+  |                |  +--^--v--+  |     |   |
|   |   +------^--v----+                +-----^--v-----+     |   |
|   |          ^  v                           ^  v           |   |
|   |   +------^--v>>>>>>>>>>>>>>>>>>>>>>>>>>>^--v-----+     |   |
|   |   |      ^  OVS-DPDK / VPP-memif vSwitch   v     |     |   |
|   |   +------^---------------------------------v-----+     |   |
|   |   |      ^           PMD Driver            v     |     |   |
|   |   +------^---------------------------------v-----+     |   |
|   |          ^                                 v           |   |
|   +----------^---------------------------------v-----------+   |
|              ^                                 v               |
|   +----------^---------------------------------v---------+     |
|   |          ^              40G NIC            v         |     |
|   |   +------^-------+                +--------v-----+   |     |
+---|---|    Port 0    |----------------|    Port 1    |---|-----+
    |   +------^-------+                +--------v-----+   |
    +----------^---------------------------------v---------+
        +------^-------+                +--------v-----+
+-------|    Port 0    |----------------|    Port 1    |---------+
|       +------^-------+                +--------v-----+         |
|                  Traffic Generator (TRex)                      |
|                                                                |
+----------------------------------------------------------------+
```

Figure 18: Multi-pod Benchmarking Scenario

C.2. Hardware Configurations

Node Name	Specification	Description
Conatiner Control for Master	- Intel(R) Core(TM) E5-2620v3 @ 2.40GHz (1socket x 12Cores) - MEM 32GB	Container Deployment and Network Allocation - ubuntu 18.04 - Kubernetes Master

	- DISK 1TB - NIC: Control plane: 1G - OS: CentOS Linux7.9	- CNI Controller - MULTUS CNI - DPDK-OVS/VPP-memif
Conatiner Service for Worker	- Intel(R) Xeon(R) Gold 6148 @ 2.40GHz (2socket X 40Cores) - MEM 256GB - DISK 2TB - NIC - Control plane: 1G - Data plane: XL710-qda2 (1NIC 2PORT- 40Gb) - OS: CentOS Linux 7.9	- Container dpdk-L2fwd - Kubernetes Worker - CNI Agent - Multus CNI - DPDK-OVS/VPP-memif
Packet Generator	- Intel(R) Xeon(R) Gold 6148 @ 2.4Ghz (2Socket X 40Core) - MEM 256GB - DISK 2TB - NIC - Data plane: XL710-qda2 (1NIC 2PORT - 40Gb) - OS: CentOS Lunix 7.9	Packet Generator - Installed Trex v2.92

Figure 19: Hardware Configurations for Multi-pod Benchmarking

For installations and configurations of CNIs, we used userspace-cni network plugin. Among this CNI, multus provides to create multiple interfaces for each pod. Both OVS-DPDK and VPP-memif bypass kernel with DPDK PMD driver. For CPU isolation and NUMA allocation, we used Intel CMK with exclusive mode. Since Trex generator is upgraded to the new version, we used the latest version of Trex.

C.3. NUMA Allocation Scenario

To analyze benchmarking impacts of different NUMA allocation, we set 6 scenarios depending on CPU location allocating to two pods and vSwich. For this scenario, we did not consider cross-NUMA case, which allocates CPUs to pod or switch in a manner that two cores are located in different NUMA nodes. 6 scenarios we considered are listed in Table 1. Note that, NIC is attached to the NUMA1.

Scenario #	vSwtich	pod1	pod2
S1	NUMA1	NUMA0	NUMA0
S2	NUMA1	NUMA1	NUMA1
S3	NUMA0	NUMA0	NUMA0
S4	NUMA0	NUMA1	NUMA1
S5	NUMA1	NUMA1	NUMA0
S6	NUMA0	NUMA0	NUMA1

Table 1: NUMA Allocation Scenarios

C.4. Traffic Generator Configurations

For multi-pod benchmarking, we discovered Non Drop Rate (NDR) with binary search algorithm. In Trex, it supports command to discover NDR for each testing. Also, we test for different ethernet frame sizes from 64bytes to 1518bytes. For running Trex, we used command as follows;

```
./ndr --stl --port 0 1 -v --profile stl/bench.py --prof-tun size=x --
opt-bin-search
```

C.5. Benchmark Results and Trouble-shootings

As the benchmarking results, Table 2 shows packet loss ratio using 1518 bytes packet in OVS-DPDK/vpp-memif. From that result, we can say that the vpp-memif has better performance that OVS-DPDK, which is came from the difference in the way to forward packets between vswitch and pod. Also, the impact of NUMA is bigger when vswitch and both pods are located in the same node than when allocating CPU to the node where NIC is attached.

Networking Model	S1	S2	S3	S4	S5	S6
OVS-DPDK	21.29	13.17	6.32	19.76	12.43	6.38
vpp-memif	59.96	34.17	45.13	57.1	33.47	44.92

Table 2: Multi-pod Benchmarking Results (% of Line Rate)

Authors' Addresses

Kyoungjae Sun
ETRI
218, Gajeong-ro, Yuseung-gu
Dajeon
34065
Republic of Korea
Phone: +82 10 3643 5627
Email: kjsun@etri.re.kr

Hyunsik Yang
KT
KT Research Center 151
Taebong-ro, Seocho-gu
Seoul
06763
Republic of Korea
Phone: +82 10 9005 7439
Email: yangun@dcn.ssu.ac.kr

Jangwon Lee
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul
06978
Republic of Korea
Phone: +82 10 7448 4664
Email: jangwon.lee@dcn.ssu.ac.kr

Tran Minh Ngoc
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul
06978
Republic of Korea
Phone: +82 2 820 0841
Email: mipearlska1307@dcn.ssu.ac.kr

Younghan Kim
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul
06978
Republic of Korea
Phone: +82 10 2691 0904
Email: younghak@ssu.ac.kr